VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘÍZENÍ POHYBU ROBOTA PODLE OBRAZOVÝCH DAT Z KAMERY

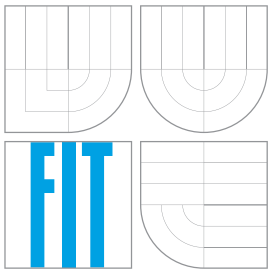
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV UHLÍŘ

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ŘÍZENÍ POHYBU ROBOTA PODLE OBRAZOVÝCH DAT Z KAMERY

CONTROLLING OF ROBOT MOVEMENTS BY ON-BOARD CAMERA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VÁCLAV UHLÍŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN

BRNO 2010

Abstrakt

Tato studentská práce pojednává o základním zpracování obrazu a jeho následném využití automatickými systémy, zejména pro využití při orientaci v prostoru a detekci překážek v robotice. Dále je v této práci navržena implementace základního rozpoznávání překážek a popis implementace pro konkrétní aplikaci na robotu Surveyor SRV-1.

Abstract

This student paper discusses the basic image processing and it's use in automatic systems, particularly for use in space orientation and obstacle detection for robots. Additionally, the work show concept implementation of the basic obstacles recognition and description of actual implementation for specific application an the robot Surveyor SRV-1.

Klíčová slova

Robotika, zpracování obrazu, detekce překážek, Surveyor.

Keywords

Robotics, image processing, obstacle detection, Surveyor.

Citace

Václav Uhlíř: Řízení pohybu robota podle obrazových dat z kamery, bakalářská práce, Brno, FIT VUT v Brně, 2010

Řízení pohybu robota podle obrazových dat z kamery

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana.

.....
Václav Uhlíř
19. května 2010

Poděkování

Tímto bych chtěl poděkovat Ing. Jaroslavu Rozmanovi za odbornou pomoc, věnovaný čas a umožnění přístupu k robotu SRV-1.

Obsah

Kapitola 1

Úvod

1.1 Roboti a robotika

Původ slova robot je odvozen od slova „robota“ a pochází z roku 1920 ze hry Karla Čapka R.U.R. Robot je tedy stroj, který má za úkol vykonávat nějakou činnost. V dnešní době se nejčastěji používají roboti jako zástupci lidské práce a to zejména jako:

- práce, u kterých je potřeba rychlost (například kuchyňský robot),
- práce, u kterých je potřeba přesnost (například laboratorní robot),
- práce, u kterých je potřeba nadlidská síla (robot ve výrobní lince).

Stále více se však objevují i roboti u kterých je potřeba nějaká vlastní inteligence nebo alespoň nějaké základní povědomí o okolí. Ať už jsou to roboti využíváni k záchranným pracím (např. prohledávání suti po zemětřesení), výzkumným pracím (podmořští roboti, vesmírní roboti) nebo roboti ve vojenské oblasti, potřebují tito roboti poznat některé prvky z okolí aby se mohli v daném prostoru orientovat. Z pohledu člověka je pro orientaci v okolí nejpřirozenější zrak, avšak pro roboty zpracování obrazu není jednoduché a tak v této oblasti nachází velké využití například infračervená čidla nebo sonary. Každý systém a čidlo má svoje klady, zápory a hlavně omezení (zpracování obrazu ve tmě, sonar ve vzduchoprázdnu, infračervené čidlo v citlivém prostředí).

Kapitola 2

Pohled kamerou robota

Chceme-li v obraze rozpoznat vzdálenost nebo polohu objektů, musíme nejdříve nějak oddělit jednotlivé předměty od zbytku obrazu a dalších předmětů. Je tedy potřeba aplikovat nějaký algoritmus, který zpracuje obraz z kamery a jehož výstupem budou data dostačující k segmentaci pro daný účel.

2.1 Stacionární kamera

Nachází-li se robot s kamerou stále na stejném místě nebo používá-li data ze stacionární kamery, můžeme k rozpoznání nových předmětů využít například „odečítání“ obrazu (Frame Differencing)[9].

Princip je celkem jednoduchý, je potřeba jeden původní snímek prostředí a porovnáváme ho s aktuálními snímky. Tam, kde se obraz výrazně liší, se s velkou pravděpodobností nachází nový objekt nebo naopak nějaký objekt chybí. Pokud dochází jen k malým osamoceným výchylkám, je to pravděpodobně vlivem šumu nebo drobnou změnou pozadí (např. třepetání se listů na stromě v pozadí nebo odlesky na hladině vody) a tyto anomálie je potřeba správně rozpoznat a případně ignorovat. Dále je vhodné čas od času aktualizovat původní snímek, který používáme jako předlohu k porovnávání. Jinak by mohlo docházet k falešnému označení prostoru například kvůli změně stínu vlivem otáčení se země. Konkrétní čas, po který se dá snímek s předlohou využít, je závislý na prostředí.

Obrázek 2.1: Odečítání obrazu (Frame differencing)

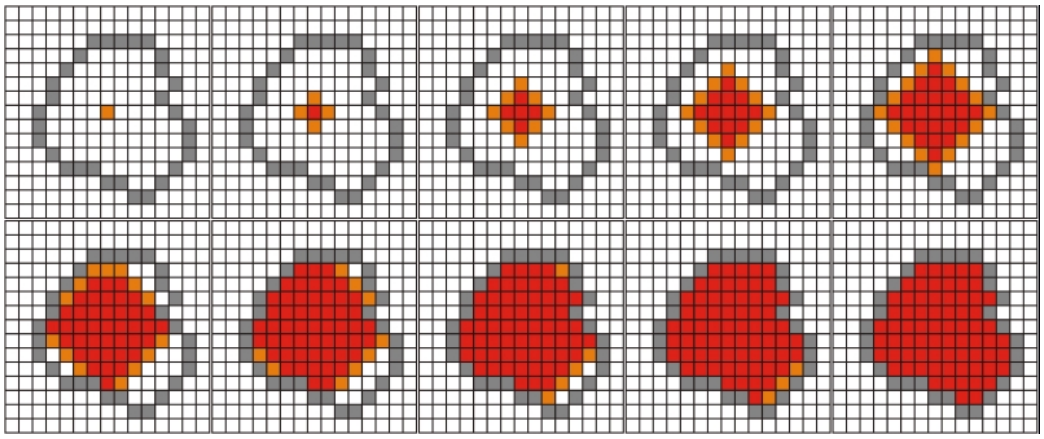
Použijeme-li místo jednoho snímku průměrování určitého počtu předchozích snímků před každým vyhodnocovaným snímkem, dosáhneme snížení rušivých elementů. Tímto způsobem získáme takzvanou „mapu aktivit“ – tedy oblasti, kde se zrovna něco změnilo. Sledováním těchto oblastí můžeme následně poznat, co se v dané oblasti děje, případně jakým směrem se detekovaný objekt pohybuje. Tato metoda je často používaná u zabezpečovacích systémů.

2.2 Nestacionární kamera

Pohybuje-li se však robot i s kamerou skrz prostředí, dostává z kamery velké množství dat, které jsou pro něj bez zpracování jen velkou hromadou pixelů. Člověk, který se zadívá do krajiny většinou automaticky pozná, co je zem, co je pozadí a co jsou nějaké předměty nebo překážky. Na robotovi je potřeba nějak tyto rozpoznávací procesy odsimulovat a data správně vyhodnotit. Kvůli této potřebě jsou tedy neustále vyvíjeny algoritmy a postupy, kterými jsou vyhledávány spojité plochy nebo přímo jednotlivé předměty. Nejznámější algoritmy jsou asi Semínkové vyplňování, Zaplavovací algoritmy nebo třeba Algoritmus rozpoznávání hran.

2.2.1 Semínkové vyplňování

Semínkové vyplňování (Flood fill) je algoritmus většinou pracující na základě rekurze a slouží k vyhledávání propojených částí obrazu na základě podobnosti jednotlivých pixelů[2]. Algoritmus funguje na principu „rozrůstajících se“ semínek. Jednotlivé pixely se stávají prvky pole, do kterého se vkládají semínka.



Obrázek 2.2: Princip čtyřsměrového semínkového vyplňování s ukládáním do fronty

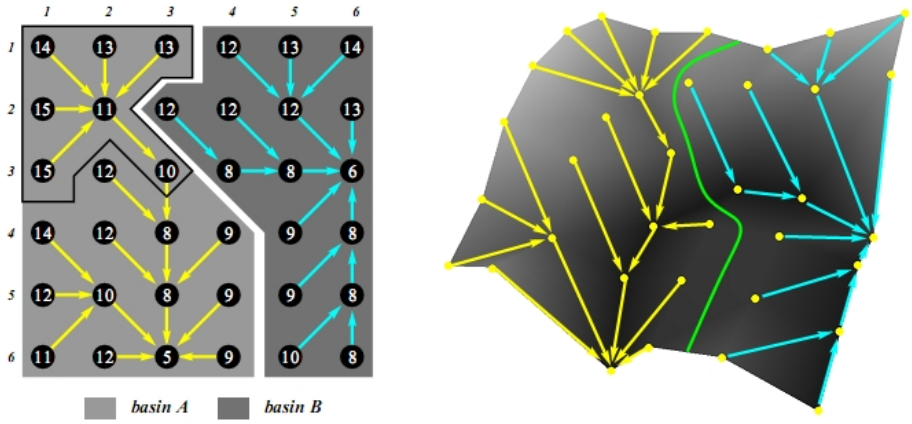
- Do obrazu bylo vloženo „semínko“.
- Do všech okolních pixelů, které vyhovují podmínkám porovnání, je vloženo semínko.
- Algoritmus opakuje tyto kroky.

Dále v žádném políčku „neroste“ semínko dvakrát (většinou je to zajištěno změnou barvy nebo polem, do kterého se označují použité pixely, takzvané „Mapy obrazu“). Díky tomuto postupu se označí plocha stejné (nebo podobné) barvy. Druhá varianta semínkového vyplňování používá k porovnání místo barvy z počátečního bodu, barvu z bodu odkud se

k tomuto bodu algoritmus dostal. Tedy algoritmus je schopen přecházet i přes postupnou změnu barvy (pokud je rozdíl sousedních pixelů vždy v mezi). Díky této změně je možné lépe označovat předměty, ale při nezaostřeném předmětu nebo nekvalitní kameře s velkým šumem může docházet k „přetékání“ algoritmu do míst mimo předmět.

2.2.2 Zaplavovací algoritmus

Zaplavovací algoritmus (Watershed Algorithm) [4] pojme obrázek jako geografickou mapu, kde čáry jsou nahrazeny „pohořím“ a souvislé plochy „údolím“. Poté je na obrázku simulováno zaplavení čímž jsou vytvořena jednotlivá jezera oddělená pohořími. Tyto „jezera“ pak reprezentují jednotlivé objekty a jejich hranice jsou hranicemi objektů.



Obrázek 2.3: Zaplavovací algoritmus na principu simulace „deště“ (převzato z [11])

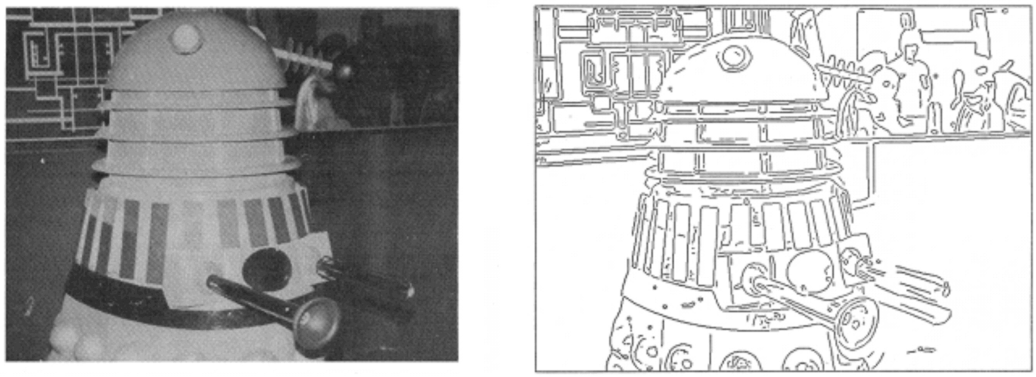
Pro správné fungování Zaplavovacího algoritmu je vhodné označit, které plochy patří ke stejnému objektu. To většinou vyžaduje zásah člověka nebo některý další rozpoznávací algoritmus.

2.2.3 Rozpoznávání hran

Rozpoznávání hran je v podstatě vyhledávání skoků v datech obrazu. Nejčastěji se k tomuto hledání využívají matice, například Sobelova matice pro detekci vertikálních změn vypadá takto:

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.1}$$

Matice se aplikuje na každý pixel obrazu, tedy v každém pixelu bude po aplikování této matice poměrový součet pixelů z vyšší řady a od něj odečten poměrový součet pixelů z nižší řady. Bude-li například obsah horních pixelů stejný jako obsah dolních pixelů bude výsledkem 0. Budou-li však řady rozdílné, uloží se do pixelu rozdíl (většinou absolutní hodnota rozdílu). Aplikováním této matice nám tedy vznikne mapa vertikálních změn, ke které se pak například pro zpřesnění ještě přičte mapa horizontálních změn.



Obrázek 2.4: Detekce hran s využitím Cannyho 6 směrových matic (převzato z [5])

Tímto způsobem se ve výsledcích objeví nejenom okraje předmětů, ale také okraje stínů od předmětů, okraje barevných rozdílů na předmětech a zlomy na 3D objektech. Z toho vyplývá problém, jak určit ty správné hrany. K tomuto účelu se u pokročilejších algoritmů využívá hystereze. Algoritmus nejprve najde nejvýznamnější hrany a pak hledá, kam tyto hrany pokračují.

Kapitola 3

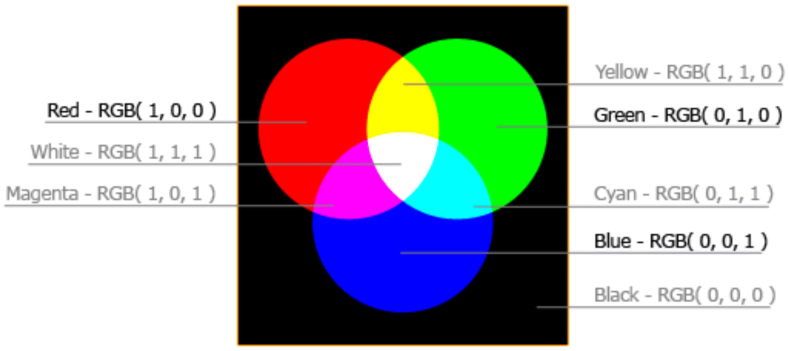
Barvy z pohledu robota

Jak již bylo řečeno, z pohledu robota je obraz jen velké množství dat, které je potřeba nějak zpracovat. Mnoho algoritmů si „vystačí“ jen s omezeným počtem barev, například odstíny šedé, které získávají buď přímo z černobílé kamery nebo transformací barevného obrazu. Ač mnoho algoritmů umí dávat dobré výsledky z černobílých dat, je použitím jen jasu barev určitá hodnota informací zahazována. Oproti tomu práce algoritmu, který barev využívá, bývá výpočetně náročnější a v robotice je často výpočetní náročnost velmi důležitým faktorem.

Při práci s barvami ve výpočetní technice je také důležité vhodně zvolit barevný model. Barevný model určuje, v kolika kanálech se barvy ukládají a jaké informace jsou v kterém kanálu. Příkladem různých typů barevných modelů jsou například RGB model nebo YUV model.

3.1 RGB model

RGB je standardní barevný model, který se využívá k zobrazování dat na monitorech a při většině manipulací s obrázky a videem v počítači. Zkratka RGB značí tři kanály: červená (Red), zelená (Green) a modrá (Blue).



Obrázek 3.1: RGB model (převzato z [7])

RGB model využívá takzvané „aditivní“ míchání barev, tedy jednotlivé kanály si můžeme představit jako jednotlivé světelné zdroje o vlnové délce odpovídající jednotlivým barvám.

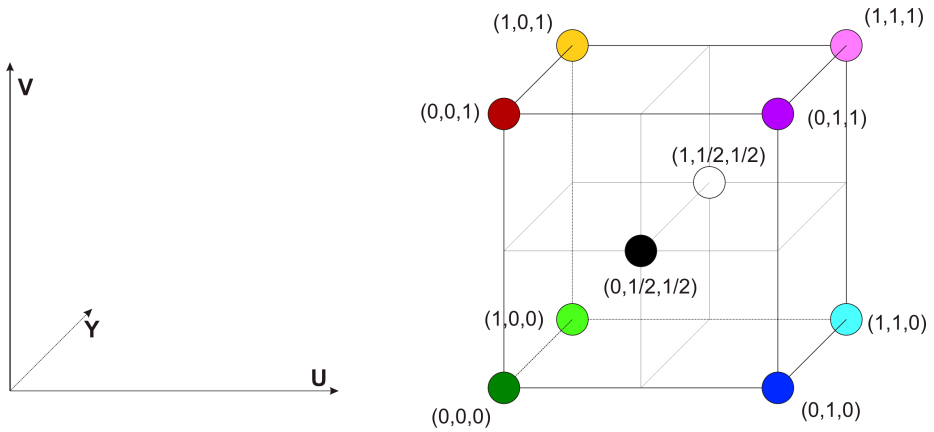
Jednotlivé hodnoty kanálů pak určují intenzitu konkrétního zdroje, například pokud budou všechny kanály mít minimální hodnotu, dostaneme černou barvu a naopak, pokud budou všechny kanály na maximu, dostaneme barvu bílou.

3.2 YUV model

YUV je barevný model využívající tři kanály. Kanály U a V určují barvu a kanál Y určuje její jas. Převod mezi RGB a YUV se dá provést například pomocí matic [6] :

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \tag{3.1}$$

YUV model vznikl při začátku barevných televizorů, kde bylo potřeba zachovat původní černobílý obraz, ze kterého se tím stala jasová složka a byly přidány dva kanály určující barvu. Díky tomu, že YUV vede jasový kanál zvlášť, je možné využít některé postupy zpracování obrazu jako pro černobílé data a zbývající barevné kanály použít k upřesnění výsledků.



Obrázek 3.2: YUV model

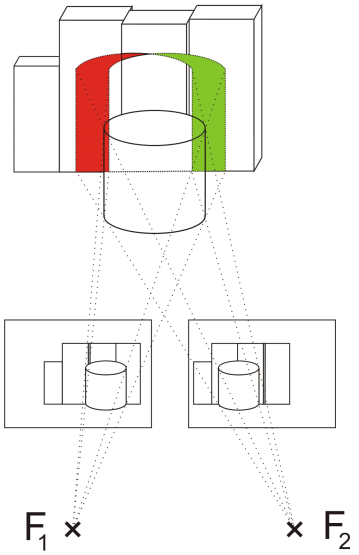
Kapitola 4

Vzdálenost předmětu

Pro bezpečný pohyb robota v prostoru nestačí jen segmentace obrazu na jednotlivé předměty. Z pohledu robota se po segmentaci objeví jen data, která říkají: „tahle oblast patří jednomu předmětu“. Aby mohl robot na předmět reagovat (přijet k němu, objet ho nebo třeba s ním interagovat), musí zjistit jeho polohu k sobě samému, případně k dalším objektům. Na zjištění vzdálenosti předmětů jsou obecně vhodnější čidla na principu měření odezvy ultrazvuku nebo laserové dálkoměry. Tato čidla však nemusí být vždy z nějakých důvodů možná. Naštěstí vzdálenosti předmětů se dají zjišťovat i ze samotných obrazových dat. Jedny z nejvýznamnějších možností jsou například stereovize nebo znalostní databáze.

4.1 Stereovize

Stereovize, jak už název napovídá, využívá rozdílu obrazu ze dvou kamer k výpočtu vzdáleností předmětů. Robot se stereovizí potřebuje vzít 2 snímky z kamer umístěných vedle sebe a namířených jedním směrem, nasegmentovat je stejným způsobem a poté najít shodné elementy. Podle posunu na jednotlivých snímcích poté lze vypočítat aktuální vzdálenost.



Obrázek 4.1: Rozdíl z pohledu robota při stereovizi

4.2 Znalostní databáze

Znalostní databáze znamená využívání předem definovaných dat. Robot je vybaven „znalostmi“ jak předměty vypadají a jak jsou velké. Objeví-li se tedy před robotem nějaký předmět, robot vyhledá v databázi, jak je ve skutečnosti velký a podle toho určí, jak je předmět daleko. Pro výpočet vzdálenosti můžeme použít vzorec vycházející z přímé úměry:

$$d = \frac{s_{db}}{s} * d_{db}, \tag{4.1}$$

kde: d značí výslednou vzdálenost, d_{db} vzdálenost v databázi, s aktuální velikost a s_{db} velikost v databázi. Například víme-li, že předmět x zabere ve vzdálenosti 10m 10 pixelů a aktuálně zabírá jen 5 pixelů, tak $d = \frac{10pix}{5pix} * 10m$ což je tedy vzdálenost 20m.

Tato metoda je vhodná například do prostředí s omezeným počtem předmětů (robot ve skladu, monitorovací systém dálnice,...). Pro obecné využití je tato metoda náročná vzhledem k potřebě velkého množství vzorů v databázi.

4.3 Alternativní metody

Na určení vzdálenosti existuje i mnoho metod, které mohou být velmi výkonné pro určité prostředí, ale pro jiné zcela nevhodné.

4.3.1 Počet pixelů

Jedna z těchto metod je například vzdálenost v pixelech od spodního okraje k předmětu. Za předpokladu, že robot jede po rovném stejnobarevném podkladu, dá se takto velmi snadno zjistit, jak daleko má k předmětu. Stačí vědět, jak vysoko je kamera a jak má velký záběr. Avšak pokud není podklad rovný nebo jsou na něm nějaké rušivé grafické prvky, stává se tato metoda velmi nepřesnou, ne-li nepoužitelnou.

4.3.2 Ostření

Další metodou je například metoda ostření, aneb robot mění ohniskovou vzdálenost a poté hledá v obraze hrany. Ve chvíli, kdy jsou hrany nejostřejší, může robot určit vzdálenost. Tato metoda má nevýhody, že robot by se během ostření neměl pohybovat a je potřeba kamera s ostřicím mechanizmem. Při ostření je navíc potřeba neustále zpracovávat relativně velké množství dat.

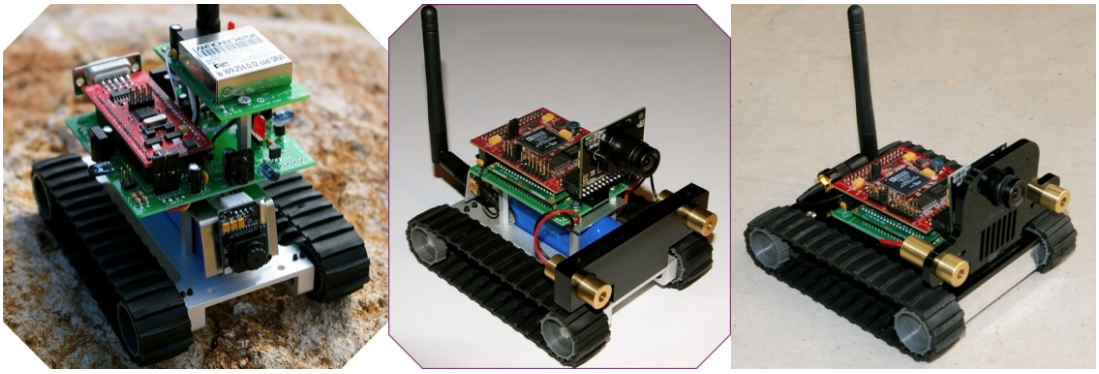
4.3.3 Poměrová vzdálenost

Robot si pamatuje scénu, kterou „viděl“ před chvílí a vypočítává vzdálenost podle poměru změny velikostí nebo pozice předmětů. (Podrobně popsáno v kapitole 6.2)

Kapitola 5

Robot surveyor SRV-1

Robot SRV-1 (plným názvem Surveyor SRV-1 Blackfin Robot) je vyvíjen firmou Surveyor Corporation v Californii. Jeho komerční historie začíná někdy kolem února roku 2006 a ke své dnešní podobě se nejvíce začal přibližovat od podzimu roku 2007, kdy přešel na řídící jednotku od firmy Blackfin a objevily se na něm lasery.



Obrázek 5.1: Vývoj robota SRV-1 (převzato z [12])

5.1 Hardware

Údaje převzaté z [12]:

- Procesor: ADSP-BF537 1000mips 500MHz , 32MB SDRAM, 4MB Flash, JTAG [1].
- Kamera: Omnivision OV9655 1.3 megapixelu, rozlišení od 160x128 do 1280x1024 .
- Bezdrátové spojení: WiFI Lantronix Matchport 802.11b/g, dosah (udávaný výrobcem): 100m v budově, 1000m bez překážek.
- Senzory: 2 laserová ukazovátka, podpora až 4 ultrazvukových senzorů.
- Pohon: Pásy s vlastními motory.
- Rychlost: 20cm – 40cm za vteřinu.

- Kostra: hliník.
- Rozměry: Délka 120mm, šířka 100mm a výška 80mm .
- Váha: 350g
- Napájení: 7.2V 2AH Li-Po baterie.

5.2 Software

5.2.1 Firmware

Firmware robota je napsán v jazyce C pod GPL Open Source licencí. Kompilace do výsledného formátu pro robota probíhá s „GNU Toolchain for the Blackfin Processor“ [3]

5.2.2 PicoC

Přímo za běhu je robot schopen překládat a vykonávat kód psaný v jazyce PicoC [10], který je speciálně upraveným jazykem C pro využití v robotice.

Zdrojový kód jazyku PicoC má zhruba jen 3500 řádek kódu a je tedy velmi úsporný vzhledem k velikosti paměti většiny robotů. Tento jazyk umožňuje většinu důležitých konstrukcí z jazyka ISO C a poskytuje možnost snadnho dopisování speciálních vlastních funkcí.

5.2.3 Konzole

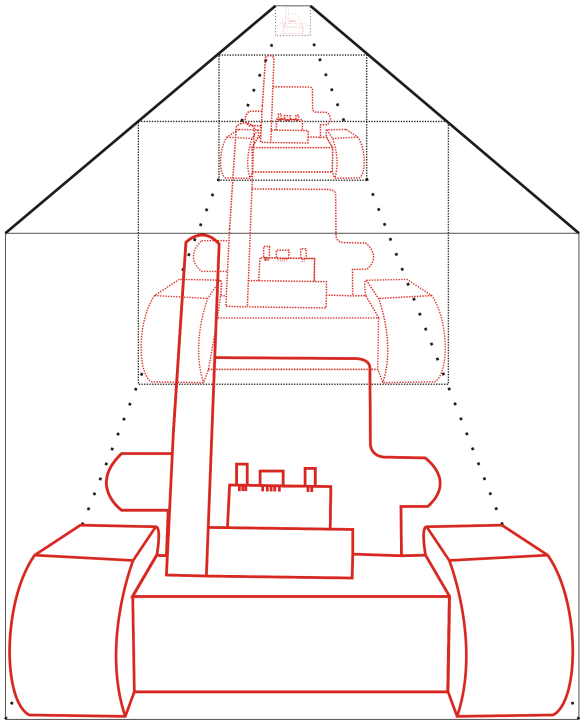
SRV-1 robot umožňuje i podporu řady konzolí:

- SRV1Console: Grafická Java konzole umožnující definování a přidávání vlastních tlačítek.
- SRV1Test: Java konzole umožňující textový vstup.
- Google Android SRV: Konzole pro telefony a zřízení se systémem Android.
- Další konzole: RoboRealm, DelphiSRV, iPhone Console, Matlab Control Scripts a mnoho dalších.

Kapitola 6

Návrh implementace samostatného pohybu

Samostatný pohyb pro robota znamená nutnost zjišťování informací o okolí a to zejména detekci překážek a vyhodnocení jejich vlivu na robota. Robot tedy musí zjistit, kde se překážky nacházejí a jak jsou daleko, aby věděl kudy může prostorem projet. Pro komplexnost problému jsem se rozhodl zaměřit tuto práci spíše na průjezd prostorem a vyhýbání se případným menším statickým překážkám. Tedy robot předpokládá, že začíná ve stavu, kdy bezprostředně před ním není překážka a bude detekovat překážky k objetí spíše něž samotné vyhledávání cesty prostorem. Ideální cestu pro robota si můžeme představit jako nějaký „tunel“, ve kterém se nesmí nacházet žádná překážka.



Obrázek 6.1: Znázornění ideální trasy pro robota

Pro implementaci samostatného pohybu robota tedy rozdělíme algoritmus na tři části:

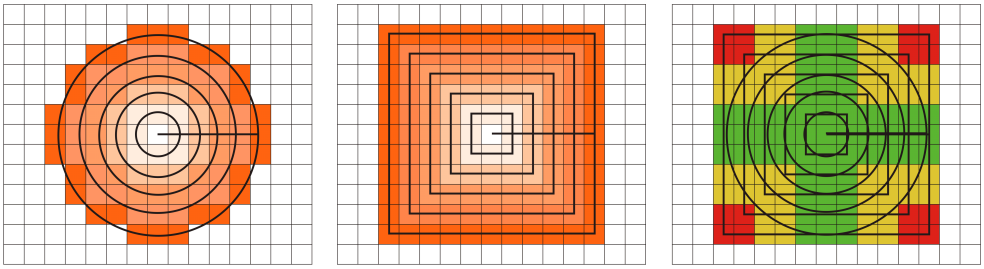
- Segmentace obrazu.
- Určení vzdálenosti.
- Vyhodnocení cesty.

6.1 Segmentace obrazu

Segmentace obrazu pro implementaci samostatného průjezdu prostorem bude potřebovat být zaměřena na detekci předmětů nacházejících se před robotem. Vzhledem k předpokladu, že robot bude začínat na prázdném úseku, je možné očekávat potenciální překážky jen v omezené zóně před robotem - u středu snímaného obrazu z kamery.

Pro segmentaci tedy bude primárně nejdůležitější detekovat nové překážky nacházející se uprostřed snímaného obrazu nebo zasahující k jeho středu. Překážky ve větší blízkosti robota se však mohou nacházet dále od středu snímaného obrazu a stále zasahovat do dráhy robota. Tyto překážky už by však měly být detekované předem ve vzdálenější pozici a v bližší pozici už bude potřeba jen určit, jak daleko vzhledem k robotu překážky jsou a jak je objet.

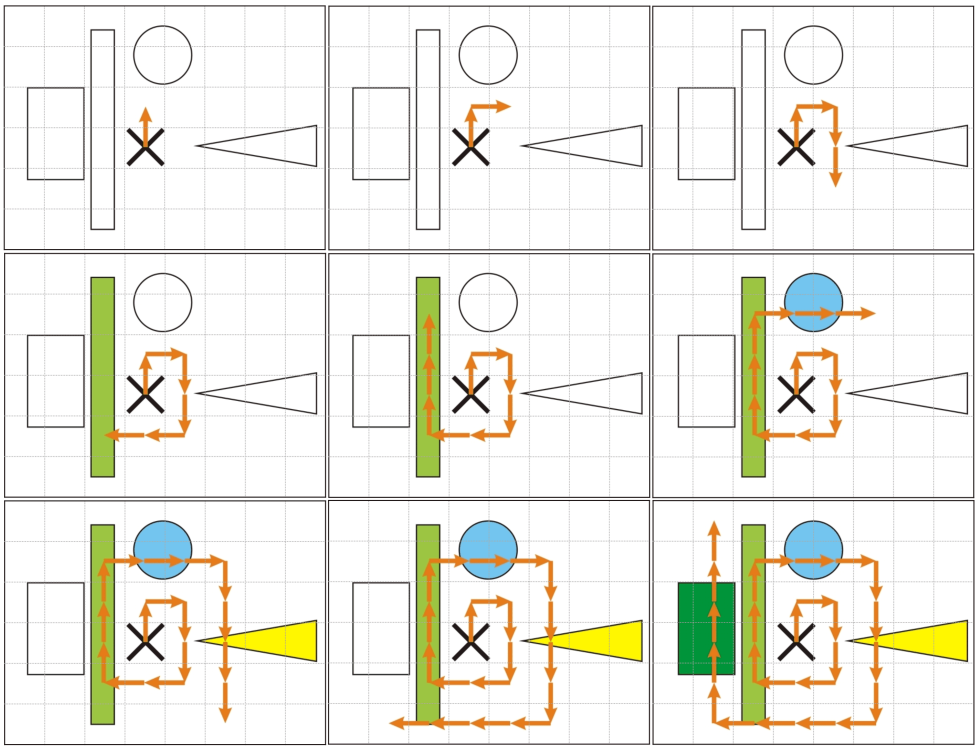
Při implementaci samotné segmentace bude pravděpodobně nejvýhodnější nějaký průchod obrazem směrem od středu ke krajům spojený s detekcí předmětů. Tento účel by mohl splnit průchod obrazem po spirále. Tedy algoritmus by začal ve středu obrazu a poté se „vydal“ do jednoho ze směrů a obcházel střed po stále větších kružnicích. Dá se však předpokládat, že vzhledem k praktické implementaci bude jednoduší procházet obraz po postupně se zvětšujících čtvercích aniž by to zbytečně výrazněji zatížilo algoritmus.



Obrázek 6.2: Znázornění průchodu kruhem, čtvercem a jejich rozdíly při pěti fázích

Pokud bude algoritmus takto procházet snímaný obraz, bude sice detekovat nějaké segmenty, ale nepodaří se mu najít, kde předměty končí dokud nebudou celé zahrnuty v oblasti projeté spirálou. To by u předmětů zasahujících ke kraji snímaného obrazu nebo u předmětů přesahující snímaný obraz znamenalo nutnost projet spirálou celý obraz a průchod od středu by tedy ztratil smysl.

Aby dokázal algoritmus najít celé předměty ve středu a jeho blízkosti, bude efektivnější při nalezení části předmětu dohledat předmět celý a teprve poté pokračovat ve spirále. Takovýmto postupem se zajistí, že prvně budou nalezeny celé předměty ve středu obrazu a teprve poté předměty více u okrajů.

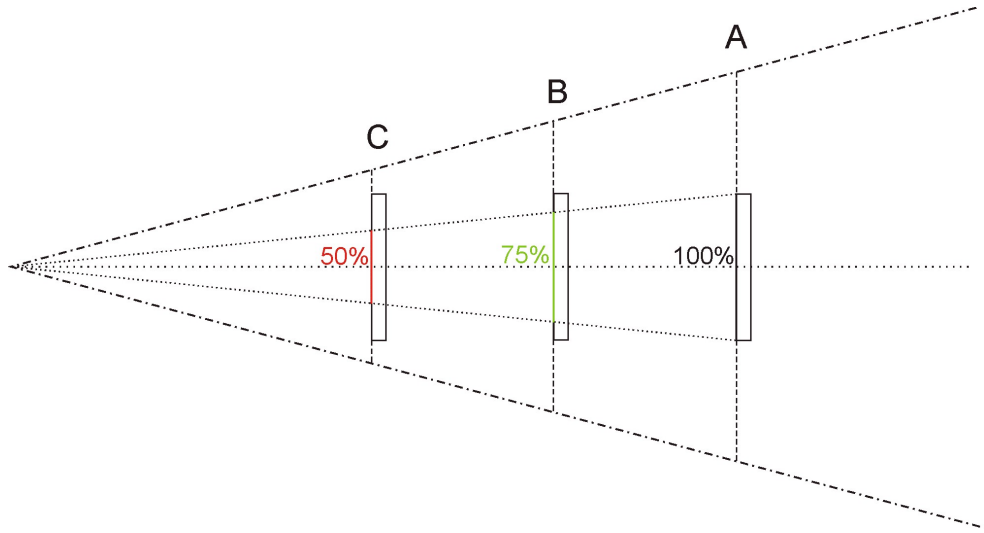


Obrázek 6.3: Znázornění segmentace obrazu

6.2 Určení vzdálenosti z obrazových dat

Určení vzdálenosti z obrazových dat je pro samostatně pohybujícího se robota klíčové. Jak již bylo popsáno v kapitole 4, existuje relativně mnoho metod pro takovéto určení vzdálenosti. Kvůli podmínkám na robotu SRV-1 jsem se rozhodl vyzkoušet využít vztahu přímé úměry a základní detekce objektu. Například pro Stereovizi by bylo potřeba dvou kamer, na použití znalostní databáze nemá robot dostatečnou kapacitu a na použití metody ostření nemá robot možnost měnit ohniskovou vzdálenost.

U robota se předpokládá, že detekci překážek bude potřebovat ve chvíli, kdy se pohybuje. Tedy při pohybu vpřed se předměty k němu blíží jeho vlastní rychlostí. Pokud tedy robot pojede konstantní rychlostí, tak mu stačí správná segmentace obrazu na jednotlivé prvky před chvílí a stejná segmentace obrazu, který byl zaznamenán dál na dráze. Jednoduše řečeno robotu si stačí pamatovat obrázek, který „viděl“ před chvílí a správně ho porovnat s aktuálním. Princip je znázorněn na obrázku 6.4.



Obrázek 6.4: Znázornění pohledu kamery na objekt v různých vzdálenostech

Pokud se objekt přesune z pozice A o jednu čtvrtinu vzdálenosti ke kameře (na pozici B) bude se jevit jako širší. Poměr šířky objektu B : A bude 100:75, tedy A se bude zobrazovat o jednu čtvrtinu menší než B.

Při posunu například z A do C (na polovinu vzdálenosti) se nám původní objekt bude jevit jen ve velikosti 50% šířky nynějšího objektu. Tedy pro výpočet vzdálenosti z libovolné pozice I do libovolné pozice J můžeme používat vzorec:

$$d_J = \frac{s_I}{s_J} d_I, \tag{6.1}$$

kde je d_X vzdálenost k objektu na pozici X a s_X je zobrazená šířka objektu na pozici X. Pro výpočet tímto vzorcem však potřebujeme znát vzdálenost d_X. Daleko pravděpodobnější je však, že budeme znát překonanou vzdálenost předmětu z bodu I do bodu J, což v případě pohybující se kamery a statického objektu je vzdálenost, kterou překonala kamera. Potřebujeme tedy spočítat, kolikrát se skok IJ může zopakovat, než se předmět dostane ke kameře, čehož dosáhneme pomocí vzorce:

$$\frac{puvodni\_velikost}{rozdil\_velikosti} \Rightarrow \frac{s_I}{s_J - s_I}, \tag{6.2}$$

výsledný vzorec bude:

$$d_J = \frac{s_I}{s_J - s_I} d_{IJ}, \tag{6.3}$$

kde d_{IJ} je vzdálenost mezi pozicemi I a J. Známe-li čas mezi mezi velikostmi I a J můžeme pomocí pozměněného vzorce ve tvaru :

$$t = \frac{s_I}{s_J - s_I} t_{IJ}, \tag{6.4}$$

dostat čas t, ve kterém se těleso při konstantním pohybu dostane do kolize s kamerou.

Při snímání objektu, který se přibližuje ke kameře, spíše než šířku budeme znát objem, vzorec tedy bude:

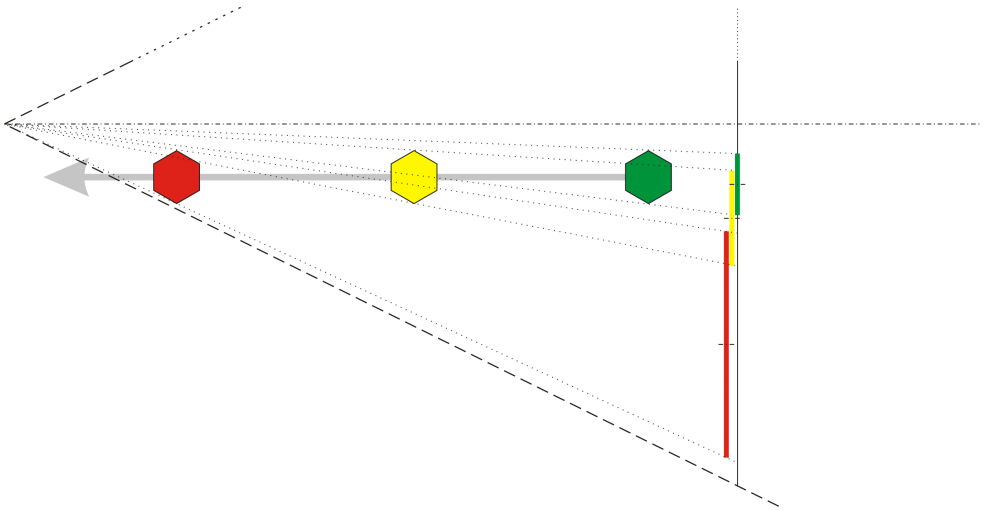
$$d_J = \frac{\sqrt{v_I}}{\sqrt{v}_J - \sqrt{v}_I} d_{IJ}, \tag{6.5}$$

z čehož po úpravě dostaneme:

$$d_J = \frac{\sqrt{v_I v_J} + v_I}{v_J - v_I} d_{IJ}, \tag{6.6}$$

6.3 Návrh řízení pohybu

Segmentací obrazu budou stanoveny směry, kterými se nacházejí jednotlivé předměty a porovnáváním velikostí bude stanovena vzdálenost. K umožnění samostatného pohybu prostorem však musíme ještě tato data interpretovat a vyhodnotit. Jak již bylo zmíněno na začátku této kapitoly, cestu robota si můžeme představit jako nějaký tunel, ve kterém se nesmějí nacházet překážky.



Obrázek 6.5: Znázornění přibližujícího se předmětu a jeho projekce

Pro znázornění pohledu na přibližující se objekty můžeme použít projekci. Na obrázku 6.5 je díky projekci vidět, že čím je objekt ke kameře blíže, tím více se na projekci zvětšuje a odchyluje se z kurzu. Odchylování se z kurzu je stejně jako velikost přímo úměrné zmenšování se vzdálenosti.

Vzdálenost předmětu od středu kurzu můžeme spočítat za pomocí znalosti vzdálenosti předmětu od kamery a poměru, při jakém se zobrazují předměty. Například víme-li, že předmět široký 5 cm se ve vzdálenosti 10 cm zobrazí jako široký 20 pixelů, můžeme spočítat reálnou šířku libovolného předmětu, známe-li jeho zobrazovací šířku a vzdálenost. Reálnou velikost předmětu můžeme tedy spočítat pomocí přímé úměry:

$$s_o = \frac{p_o}{p_{ref}} * s_{ref}, \tag{6.7}$$

kde s_o je výsledná reálná velikost, p_o je velikost zobrazovaného objektu v pixelech, p_{ref} je referenční zobrazovaná velikost v pixelech a s_{ref} je referenční reálná velikost. Tento vzorec

je však použitelný jen pokud je zkoumaný objekt ve stejné vzdálenosti jako byl referenční objekt.

Abychom mohli počítat reálnou velikost předmětu v libovolné vzdálenosti, musíme přepočítat počet pixelů na vzdálenost. Pro přepočítání zobrazovací velikosti objektu na vzdálenost referenčního objektu použijeme vzorec nepřímé úměry:

$$p_{new} = \frac{d_o}{d_{ref}} * p_o, \tag{6.8}$$

kde p_{new} je výsledná zobrazená velikost v referenční vzdálenosti d_{ref} a p_o je aktuální zobrazovaná velikost ve vzdálenosti d_o.

Kombinací vzorce 6.7 a 6.8 dostaneme vzorec pro výpočet reálné velikosti objektu v libovolné vzdálenosti:

$$s_o = \frac{\frac{d_o}{d_{ref}} * p_o}{p_{ref}} * s_{ref}, \tag{6.9}$$

kde po úpravě dostaneme:

$$s_o = \frac{d_o * p_o * s_{ref}}{d_{ref} * p_{ref}}. \tag{6.10}$$

Protože hodnoty s_{ref},d_{ref} a p_{ref} jsou referenční a neměnné, je možné je nahradit konstantou:

$$REF = \frac{s_{ref}}{d_{ref} * p_{ref}} \tag{6.11}$$

a celý výpočet pak bude probíhat jen pomocí vzorce:

$$s_o = d_o * p_o * REF. \tag{6.12}$$

Samotná reálná velikost objektu k rozlišení, zda je nebo není překážkou, nestačí. Avšak při jízdě rovnoběžně s pohledem kamery platí, že objekt, který je v centru obrazu, je přímo na dráze a tedy je překážkou a pro objekty, které jsou v okolí středu, můžeme použít vzorec 6.12, kde místo šířky objektu p_o dosadíme vzdálenost nejbližšího okraje ke středu obrazu p_{oc}. Na vzdálenost objektu od středu platí stejné vztahy jako na šířku objektu, které byly popsány výše, neboť místo mezery mezi objektem a osou pohledu si můžeme představit virtuální objekt vyplňující tuto mezeru.

Kapitola 7

Praktická implementace samostatného pohybu

Pro implementaci vlastního algoritmu jsem se rozhodl využít firmware robota SRV-1. Firmware robota poskytuje základní funkce pro práci s obrazem a měření času.

7.1 Implementace segmentace obrazu

Segmentace obrazu pro tento účel bude pracovat nejdříve s pixely od středu po čtvercové spirále směrem ke kraji. Postup po takovéto spirále se dá popsat pseudokódem jako:

```
směr=0; i=0; j=0
pozice.x=img.width/2
pozice.y=img.height/2
for x:
  i++
  if(i>=j/2)j++; i=0; ++směr%=4
  switch(směr):
      case 0: pozice.x--
      case 1: pozice.y++
      case 2: pozice.x++
      case 3: pozice.y--
  operace_s_pixelem()
```

Kód 7.1: Průchod obrazem po spirále

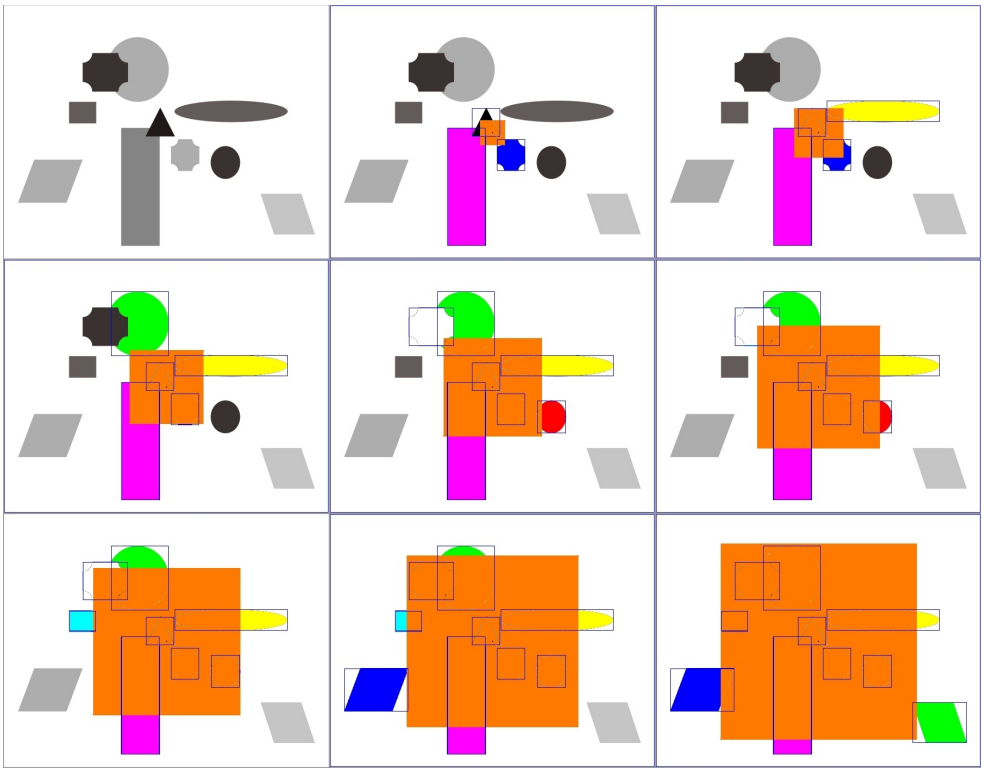
Algoritmus se díky tomuto kódu dokáže pohybovat po spirále od středu k okrajům (nejprve postoupí o pixel výše a pak se točí po směru hodinových ručiček, kdy vždy po dvou rovných úsecích zvětší velikost procházeného úseku). U tohoto algoritmu je však nutné hlídat počet segmentů, protože překročí-li počet segmentů dvojnásobek výšky, může dojít k přetečení přes okraje obrazu.

Kdykoli algoritmus narazí na pixel, který není už součástí nějakého objektu (není označen v mapě obrazu jako obsazený), spustí se z takového pixelu semínkovým algoritmem po pixelech podobné barvy dokud nenarazí na hranice objektu. Při tomto procházení objektu si označuje prošlé pixely do mapy, aby se náhodou nestaly součástí více objektů. Při procházení

se zároveň ukládá k danému objektu jeho počet pixelů, krajní hranice v osách x a y a vážený střed. Vážený střed se zde průběžně vypočítává jako:

$$R_{act} = \frac{R_{last} * C + P_{act}}{C + 1}, \tag{7.1}$$

kde R je vážený střed, C je dosavadní počet pixelů v objektu a P_{act} je aktuální pozice pixelu. Tedy k posledně spočítanému středu se poměrově přičte nová pozice. Tento střed je pak využíván v porovnání, zda se jedná o stejné objekty.



Obrázek 7.1: Výstup prototypové funkce segmentace vytvořené pomocí knihovny OpenCV

Implementace semínkového vyplňování by se pseudokódem dala popsat jako:

```
queue.push(pixel.position)
while(queue):
    position=queue.pop()
    object.add_pixel(position)
    for(directions)
        tmpPosition=position+direction
        if(map(tmpPosition)==free)
            if(color(position) <= color(tmpPosition)+color_limit)
                queue.push(tmpPosition)
                map(tmpPosition)=object.id
            else
                map(tmpPosition)=not_free_for(object.id)
```

Kód 7.2: Pseudokód optimalizovaného semínkového vyplňování

Semínkové vyplňování jsem implementoval za pomocí fronty (pro ušetření paměti). Semínkové vyplňování také využívá adaptivní barvy, tedy každý nový pixel, který se kontroluje, se porovnává s barvou pixelu, ze kterého se k tomuto pixelu algoritmus dostal. Je-li barevný rozdíl v určených mezích, je v mapě obrázku pixel označen jako součást objektu a nemůže se již stát součástí žádného jiného. Není-li rozdíl v určených mezích, je tento pixel označen jako nepatřící do konkrétního objektu a tento pixel nebude už k tomuto objektu porovnáván. Takovýto pixel se samozřejmě může později stát součástí jiného objektu.

Dostane-li se algoritmus pomocí semínkového vyplňování ke spodní hranici, je daný objekt označen jako pravděpodobný podklad a je zahozen. Stejně tomu je s objekty, které přesahují krajní hranice obrazu – jedná se buď o pozadí nebo jsou to již dříve detekované objekty, které by neměly stát v dráze.

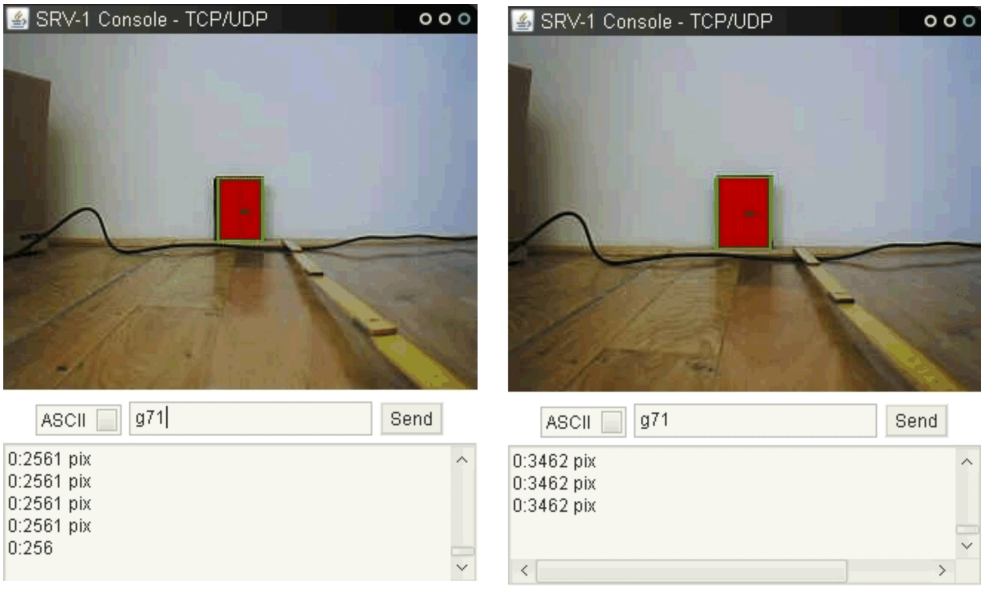
7.2 Implementace určení vzdálenosti

Implementace určení vzdálenosti je na robotu založena na opakovaném obrazovém zachycení předmětu při pohybu. Při testování jsem zjistil, že důležité faktory pro správné určení vzdálenosti jsou:

- správnost rozpoznání objektu,
- přesnost určení velikosti objektu v závislosti na tvaru, rozlišitelnosti barvy a osvětlení předmětu,
- reálné velikosti předmětu.

7.2.1 Testování přesnosti určení vzdálenosti

Pro základní testování jsem používal černý objekt (na snímku vyznačen červeně) o velikosti 9*12 centimetrů před bílým pozadím. Objekt ve vzdálenosti 80 centimetrů byl detekován



Obrázek 7.2: Testovací objekt ve vzdálenosti 80 a 70 cm od kamery

s obsahem 2561 pixelů a stejný objekt ve vzdálenosti 70 centimetrů byl detekován s obsahem 3462 pixelů. Použijeme-li tedy vzorec z kapitoly 6:

$$d_J = \frac{\sqrt{v_I v_J} + v_I}{v_J - v_I} d_{IJ} \tag{7.2}$$

do kterého dosadíme naměřené hodnoty:
$v_I = 2561px$
$v_J = 3462px$
$d_{IJ} = 10cm$

$$d_J = \frac{\sqrt{2561px * 3462px} + 2561px}{3462px - 2561px} 10cm \tag{7.3}$$

Dostaneme vzdálenost předmětu

$$d_J = 61.4718cm \tag{7.4}$$

tedy odchylka při tomto měření byla 12%.

Při druhém testu ve vzdálenostech 50 a 40 cm byl objekt detekován s obsahem 6819 a 10688 pixelů.

Obrázek 7.3: Testovací objekt ve vzdálenosti 50 a 40 cm od kamery

dosazením do vzorce dostaneme:

$$d_J = \frac{\sqrt{6819px * 10688px} + 6819px}{10688px - 6819px} 10cm \tag{7.5}$$

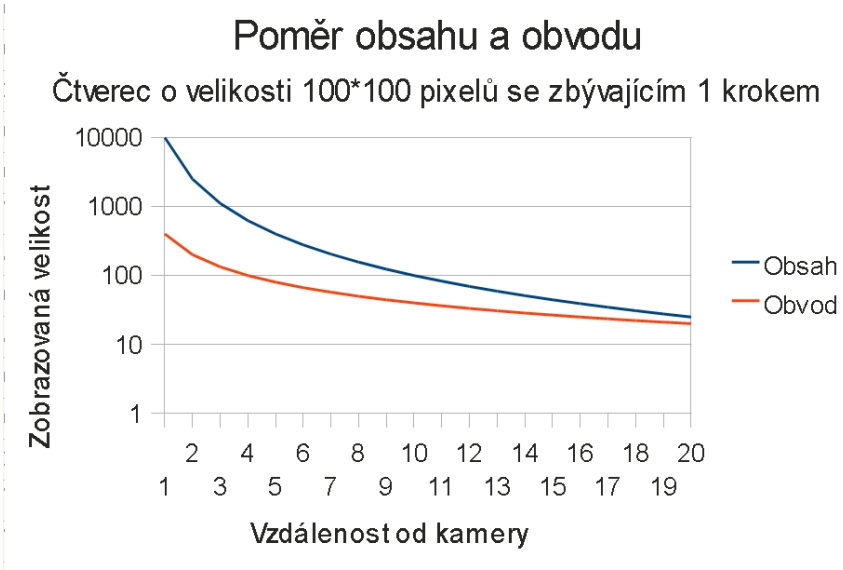
tedy

$$d_J = 39.6900cm \tag{7.6}$$

Z toho vyplývá, že odchylka u tohoto měření od skutečné vzdálenosti je 0,8%.

Jelikož k nejvíce chybám v detekci obsahových pixelů dochází na okraji předmětu, je proto odchylka závislá na poměru obsahu k obvodu, tedy objekty s menším poměrem obvod/obsah například kružnice nebo čtverce, budou detekovány přesněji, než například trojúhelník nebo dlouhý obdélník. Přesnost detekce také závisí na velikosti předmětu z pohledu kamery. Je-li předmět z pohledu kamery „jen pár pixelů“, budou chyby a tedy nepřesnost výpočtu mnohem výraznější, než když bude objekt zabírat většinu obrazu. Tyto rozdíly jsou zaviněny hlavně šumem kamery a artefakty obrazu a zvětšují se přímo úměrně barevné podobnosti předmětu s pozadím. Pro velké množství variability zobrazení předmětu se tedy nedá jednoznačně určit maximální a minimální odchylka. Avšak při testování na objektu velikosti 9*12 cm ve vzdálenosti 50 centimetrů docházelo k odchylce ± 50 pixelů z celkové velikosti 6848 pixelů obsahu v této vzdálenosti. Při výpočtu vzdálenosti tohoto objektu porovnávaného s objektem ve vzdálenosti 40 centimetrů dosahuje odchylka maxima 1,4679 centimetru, tedy 3% celkové vzdálenosti ke kameře.

Například u přibližujícího se čtverce platí, že jeho zobrazovaný objem roste nepřímo úměrně vzdálenosti, zatímco okraje působící nepřesnosti jsou odmocninou zobrazovaného obsahu. Tedy čím větší je zobrazovaná velikost čtverce, tím se snižuje nepřesnost.

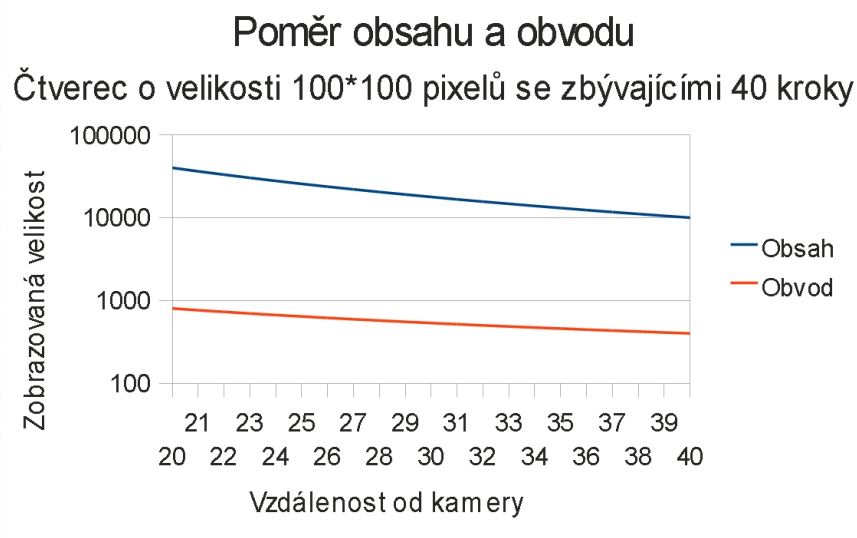


Obrázek 7.4: Graf obsahu a obvodu u čtverce o straně 100 pixelů na logaritmickém měřítku

Na grafu obrázku 7.4 je vidět, že zobrazená velikost obvodu na kroku 20 téměř dosahuje zobrazené velikosti obsahu a tedy potencionální oblast k výskytu chyb je stejná jako výpočetní plocha. Tento problém nastává, protože velikost strany v kroku 20 je 5 pixelů a obsah tedy je 25 pixelů. Segmentovací algoritmus má však omezení, že jakýkoli objekt s obsahem menším jak 200 pixelů je označen za potenciální shluk šumu nebo zbytkové plochy na hranicích větších předmětů a zahozen. Algoritmus by tedy počítal s tímto čtvercem jen ve vzdálenosti 1 až 7, kde je poměr obvodu k obsahu 28%.

Na obrázku 7.4 je graf čtverce v rozmezí 1 až 20 kroků. Tedy ve chvíli, kdy čtverec má zobrazenou velikost strany 100 pixelů, by další krok znamenal náraz do kamery. Tento

objekt je tedy hodně malý nebo je zde velký krok. Většinou jsou ale objekty algoritmem detekovány mnohem dříve a například při detekci objektu 100*100 pixelů ve vzdálenosti 40 kroků od kamery je poměr obsahu a obvodu v následujících 20 krocích v rozmezí 2 až 4 %.



Obrázek 7.5: Graf obsahu a obvodu u čtverce o straně 100 pixelů se zbývajícíma 40 kroky

Konkrétní procentuální hodnoty chyb tedy jsou závislé především na osvětlení a podobnosti barvy předmětu s barvou pozadí. Z hodnoty podílu obvodu k obsahu však můžeme vyvodit potencionální prostor se zvýšeným rizikem chyb, a proto je potřeba jej brát v úvahu.

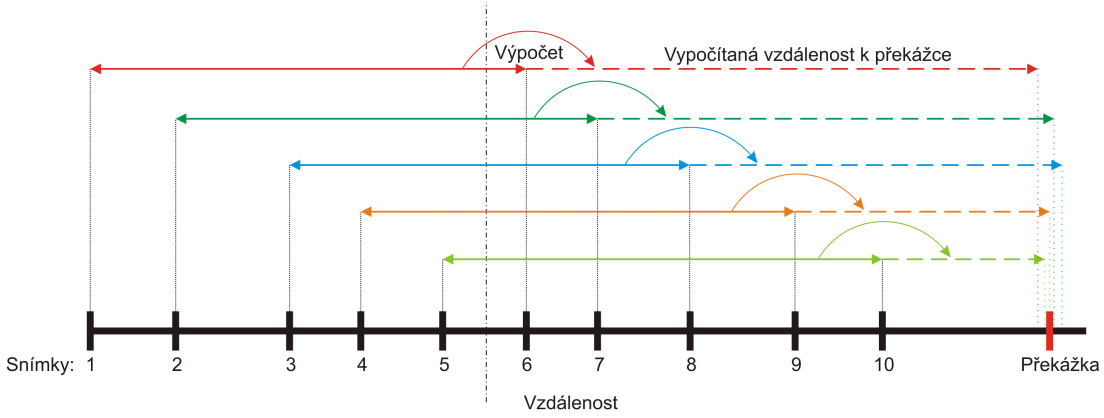
7.2.2 Výpočet vzdálenosti

Výpočet vzdálenosti počítá zbývající čas do přijetí k překážce. Pro větší přesnost se nepočítá časová vzdálenost ze snímků po sobě, ale experimentálně bylo zjištěno, že je výhodnější vypočítávat časovou vzdálenost ze snímků s rozestupem 5, což při aktuální implementaci odpovídá zhruba 600 milisekundám a při standardní rychlosti přibližně 11 centimetrů. Takto je vypočítáno 5 hodnot vzdálenosti překážky z posledních 10 snímků, znázorněno na Obrázku 7.6 . Z těchto 5 hodnot se vzdálenost k předmětu následně vypočítá jako:

```
distance=0
for(i: 1..5):
    distance-=time_diff(img[i+4],img[i+5])
    distance=((i-1)*distance+dist(i,i+5))/i
```

Kód 7.3: Pseudokód výpočtu vzdálenosti z posledních 10 hodnot

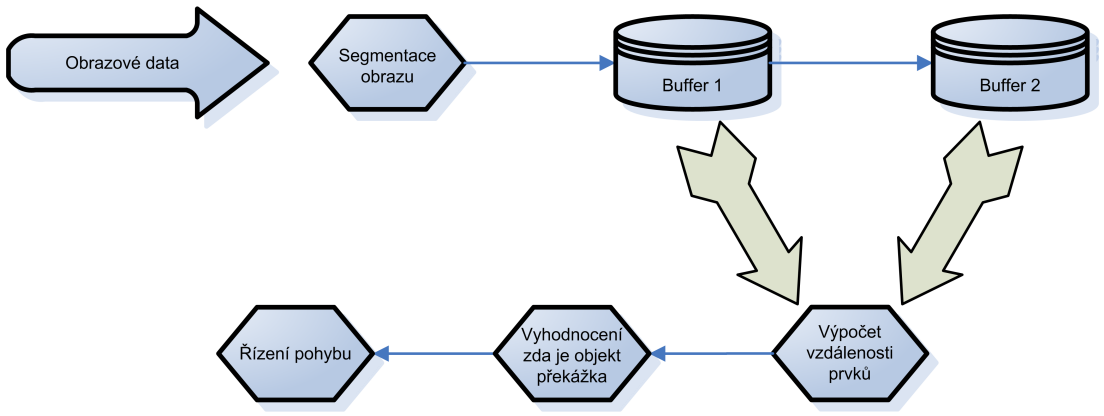
Nejprve se vždy odečte od aktuální hodnoty vzdálenost aktuálního a předchozího snímku. Poté je udělán vážený průměr dosavadních hodnot (předchozí průměr s váhou počtu snímků, ze kterých byl vypočítán a aktuální snímek s váhou 1). Po každém cyklu je výstupem v proměnné *distance* vzdálenost od snímku $i + 5$, tedy po kroku $i = 5$ bude v proměnné *distance* vzdálenost k překážce od snímku 10, který je nejaktuálnější.



Obrázek 7.6: Výpočet vzdálenosti k překážce

7.3 Implementace řízení pohybu

Implementace řízení pohybu využívá dvou předchozích částí (segmentace a výpočtu vzdálenosti) a rozhoduje o pohybu robota. Vstupní data obrazu jsou po segmentaci převedeny do reprezentace jednotlivých objektů a jejich zobrazovacích vlastností. Tyto reprezentace jsou bufferovány do dvou bufferů po pěti snímcích. Z těchto bufferů se následně vypočítává vzdálenost ke konkrétním objektům. Je-li detekovaný objekt v menší než určené vzdálenosti je řešeno, zda objekt je nebo není překážkou.



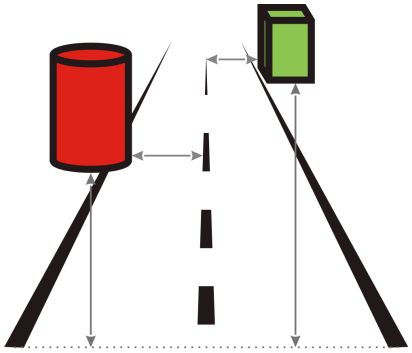
Obrázek 7.7: Schéma algoritmu řízení

7.3.1 Předpoklad kolize

Algoritmus předpokladu kolize má na starosti rozhodování, zda objekt je nebo není překážkou. Jak již bylo popsáno v kapitole 6.3, zasahuje-li objekt do středu obrazu a tedy do osy dráhy, tak je překážkou. Je-li objekt v okolí středu, tak pomocí jeho reálné vzdálenosti zobrazované vzdálenosti od středu a koeficientu REF vypočítám jeho vzdálenost od středu osy dráhy vzorcem:

$$s_o = d_o * p_o * REF. \tag{7.7}$$

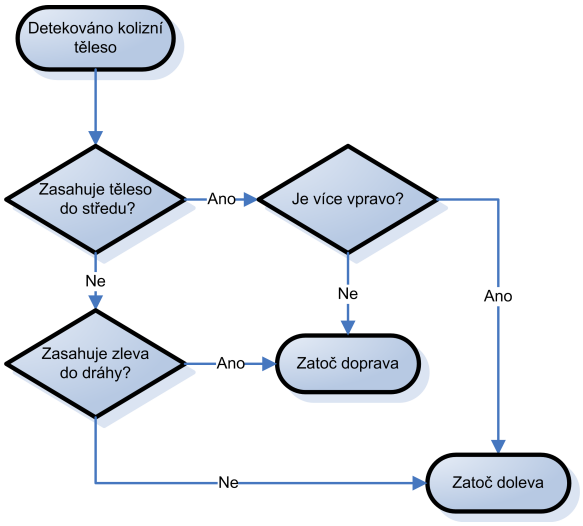
Z vypočtené vzdálenosti plyne, zda objekt leží na dráze robota, tedy zda je nebo není překážka.



Obrázek 7.8: Detekce kolize

7.3.2 Úhybný manévr

Vrátí-li algoritmus předpokladu kolize informaci, že se v dráze nachází kolizní objekt, je potřeba se objektu nějak vyhnout. Algoritmus úhybného manévru nejprve ověří, jestli objekt zasahuje do středu, a pokud ano, tak z které strany. Pokud objekt nezasahuje do středu, tak algoritmus zjistí ze které strany objekt do dráhy zasahuje a na kterou stranu má s robotem zahnout. Způsob rozhodování je znázorněn na obrázku 7.9 .



Obrázek 7.9: Rozhodovací strom úhybného algoritmu

Kapitola 8

Zhodnocení vytvořeného programu

Vytvořený program jsem testoval převážně na předmětech rozměrů 7 cm ∗ 9 cm až 12 cm ∗ 20 cm a vzdáleností od robota minimálně 50 cm. Většina základních testů, které splňovaly předpoklady, pro které byl algoritmus napsán, dopadly v pořádku dle očekávání.
Nalezené nedostatky a problémy při rozšířených testech by se daly rozdělit do kategorií:

- Problémy s velikostí předmětů.
- Problémy s barevností předmětů.
- Problémy s více překážkami a překážkami speciálních tvarů.
- Externí vlivy.

8.1 Problémy s velikostí předmětů

Dodatečné testy byly prováděny i s předměty o rozměrech 3 cm ∗ 7 cm a 24 cm ∗ 32 cm.

8.1.1 Projevy problému

Robot nedokáže správně detekovat překážky menších či větších rozměrů, než bylo stanoveno v základním předpokládu.

Malé překážky robot nedokáže detekovat z důvodu ignorace předmětů menších jak 200 pixelů implementované v segmentaci obrazu. Tyto předměty jsou často detekovány pozdě a robot do nich narazí nebo častěji je „přejede“. Ignorace malých předmětů byla v algoritmu vytvořena kvůli falešným detekcím předmětu na okrajích jiných předmětů. Například na hraně předmětu o jednom ze zobrazovaných rozměrů velikosti 100 pixelů se může zobrazit jiný odstín nebo barva kvůli odlesku okolí. Pokud tento odlesk zasahuje do řady dvou a více pixelů, může docházet k falešné detekci předmětu, který pravděpodobně bude graficky velmi nestabilní. Hranice 200 pixelů se jeví jako vhodná hranice při zaměření detekce na předměty o velikosti stanovené předpoklady využití algoritmu.

Velké překážky robot rozpozná v dálce, avšak pokud se překážky dostanou mimo zorné pole robota dříve než robot dosáhne definované vzdálenosti pro začátek úhybného manévru, tak robot tyto překážky začne ignorovat a pokračuje stále v jízdě rovně. Tyto překážky robot ignoruje, protože není schopen spočítat jejich obsah a nemůže rozhodnout o jejich aktuální vzdálenosti.

8.1.2 Navrhované řešení

Navrhované řešení pro malé překážky je zlepšení segmentace obrazu. Zlepšení segmentace pro tento účel by mohlo být dosaženo aplikováním filtru na odstranění šumu, zvýšením využitého rozlišení obrazu, případně zavedením heuristiky umožňující rozpoznat, co je nepřesnost v okolí předmětu a co je samostatný předmět.

Správného chování u velkých překážek by se dalo dosáhnout třemi možnostmi:

- včasný úhybný manévr,
- paměť detekovaných prvků,
- význačné body.

Je-li překážka detekovaná z větší vzdálenosti a hrozí, že se nějaká její část dostane mimo zorné pole robota, mohl by robot reagovat předčasným úhybným manévrem. Tento způsob by však znamenal, že se robot nemůže přiblížit k větším překážkám a například ve volném terénu by nemohl jet ve směru k jakémukoli většímu objektu na obzoru a byl by zbytečně blokován.

Překážky detekovány z větší vzdálenosti by také bylo možné zaznamenat do paměti robota a kontrolovat zpětně, zda s nimi nehrozí kolize. Tato metoda by však vytvářela další požadavky na paměť robota a jeho výkon. Dalším negativním faktorem při tomto postupu je nepřesnost detekce z velké dálky ve vztahu k měření relativním časem do srážky implementovaném na robotu. Případné chyby by do řízení tímto způsobem také zanášely předměty, na které by byl částečně blokován výhled kvůli jiným předmětům.

Přesnější detekce velkých předmětů z blízkosti by se dalo dosáhnout využitím klíčových bodů v obraze [8]. Algoritmus by provedl segmentaci na objekty a následně se pokusil najít v objektu významné klíčové body. Při nalezení takovýchto bodů je pak možno využít porovnávání vzdálenosti těchto bodů místo výpočtu ze zobrazované velikosti. Tento způsob by mohl pomoci i pro přesnější detekci dalších předmětů avšak, jeho nároky na výkon by pravděpodobně několikrát převyšovaly výkonnostní možnosti na robotu.

8.2 Problémy s barevností předmětů

Při návrhu implementace nebyly řádně specifikovány požadavky na barevnost předmětů, avšak ta se už při počátečních testech ukázala jako velmi důležitý faktor při segmentaci.

8.2.1 Projevy problému

Překážka je špatně nebo není vůbec detekovaná. Tento problém se často projevuje tak, že výstupy segmentace obrazu jsou velmi rozdílné a robot za jízdy proti překážce nedokáže přiřadit aktuálně detekované objekty k předchozím detekovaným objektům. U takovýchto objektů pak robot není schopen určit vzdálenost a tedy rozhodnout, zda objekt je nebo není překážkou. V horším případě robot překážku nevidí vůbec, protože mu splývá s pozadím nebo podložím kvůli „přetékání“ semínkového vyplňování přes hranice objektu.

V části semínkového algoritmu (během segmentovacího procesu) dochází k procházení pixelů podle vzájemné barevné vzdálenosti. Tato barevná vzdálenost je vyjádřena konstantami pro jednotlivé kanály. Účelem těchto konstant je, aby algoritmus byl schopen projít přes postupný barevný přechod v rámci jednoho předmětu. Avšak při špatném osvětlení a tedy zvýšeném šumu dochází u méně ostrých hran k falešnému přechodu a algoritmus může přetéct mimo objekt.

8.2.2 Navrhované řešení

Řešením tohoto problému by například bylo zvýšení využívaného rozlišení a tedy menší vliv šumu na hrany. Toto řešení by však znamenalo násobné zvýšení výkonnostních požadavků a výrazné zpomalení celého běhu robota.

Další možností řešení problému nejasných hranic kvůli barveným přechodům by mohla být implementace detekce hran s hysterezí popsaná v kapitole 2.2.3. V obraze by takto vznikly hranice, přes které by semínkový algoritmus nesměl přetéct.

8.3 Problémy s více překážkami a překážkami speciálních tvarů

Algoritmus podle svého návrhu v kapitole 6 se zabývá hlavně detekcí překážek. Tedy pokud se před robotem vyskytuje více objektů zároveň, je schopen rozhodnout, který je a který není překážkou, ale nedokáže najít ideální cestu. Problémy také vznikají ve chvíli, kdy je vysoká koncentrace překážek na dráze. Například vyskytnou-li se dvě vedle sebe ležící překážky zasahující do dráhy z obou stran a není mezi nimi místo na projetí, robot se v podstatě zasekne a nemůže se rozhodnout, kam jet.

Problémy u překážek složitějších tvarů se projevují nemožností robota podjet část objektu (například most nebo „semafor“) kvůli použité reprezentaci dat vycházející ze segmentace obrazu. Tento problém by se dal vyřešit detekcí kolize podle mapy objektů, která vzniká jako vedlejší produkt segmentace a není v aktuálním algoritmu využívána.

8.4 Chyby způsobené externími vlivy

Vlivy externích podnětů nebyly testovány, avšak dá se předpokládat, že mohou mít velmi významný vliv na funkčnost robota. Významnými vlivy zde mohou být například:

- Vlastnosti podloží:
  Robot pro detekci vzdálenosti využívá relativního času do srážky, pokud se na dráze změní vlastnosti podloží, může dojít ke změně rychlosti a tedy i času zbývajícího k dojezdu k předmětu.
- Světelné podmínky:
  Robot pro porovnávání shodnosti aktuálně detekovaných objektů a dříve detekovaných objektů využívá porovnávání barev. Budou-li se světelné podmínky měnit tak, že se budou i významně měnit barvy objektů, robot k nim nedokáže určit vzdálenost.

Kapitola 9

Závěr

Cílem této práce je poskytnout základní pohled na problematiku orientace v prostoru, samostatný pohyb robotů a ukázat konkrétní příklad algoritmu pro základní funkce samostatného pohybu.

Pro implementaci algoritmu jsem zvolil netradiční řešení, protože podobné problémy jsou řešeny velmi často a stále nebylo plně dosaženo chtěného výsledku. Můj návrh algoritmu a jeho následná implementace ukázaly, že jsou v určitém prostoru použitelný a při implementaci některých z navrhovaných vylepšení v kapitole 8 by mohly být využity i v širším okruhu. Například při vylepšení kolizní detekce (nyní umí robot poznat, jen jestli objekt zasahuje do dráhy v horizontální úrovni), by se takovýto algoritmus mohl využívat k detekci překážek pro letadla nebo vesmírné lodě. Vzhledem k tomu, že algoritmus počítá s relativní rychlostí do srážky, budou správně vyhodnoceny i objekty, které se konstantně pohybují směrem proti kameře nebo naopak ve směru pohledu kamery. Dále pokud by se implementovaly vektory pohybu, do kterých by se podle rovnic zmíněných v této práci dosazovaly pozice předmětů, byl by algoritmus schopen počítat s předměty pohybujícími se po libovolné konstantní dráze.

Hlavní výhody algoritmu popsaného v této práci je, že mu k vyhodnocení stačí pohled jednou kamerou a čas mezi jednotlivými snímky obrazu. Díky potřebě jen jedné kamery a žádných dalších senzorů může být celé zařízení relativně malé, levné a energeticky nenáročné, což postrádá většina jiných řešení, která se dnes používají.

Literatura

[1] Analog-Devices: ADSP-BF537: Blackfin Processor. 2009, [Online; 12-Března-2010].
URL http://www.analog.com/en/embedded-processing-dsp/blackfin/adsp-bf537/processors/product.html

[2] André, P.: Intelligent Flood Fill or: The Use of Edge Detection in Image Object Extraction. *University of Southampton Intelligent*, 2005.

[3] Blackfin-Linux: GNU Toolchain for the Blackfin Processor. 2010, [Online; 12-Března-2010].
URL http://blackfin.uclinux.org/gf/project/toolchain

[4] Bradski, G.; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[5] Canny, J.: A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, 1987: str. 184.

[6] Dupuis, É.: Optimizing YUV-RGB Color Space Conversion Using Intel's SIMD Technology. 2003.

[7] equasys: RGB Color Format. 2010, [Online; 25-Dubna-2010].
URL http://www.equasys.de/colorformat.html

[8] Lowe, D.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, ročník 60, č. 2, 2004: s. 91–110.

[9] Rajagopalan, A.; Chellappa, R.: Vehicle detection and tracking in video. In *Proc. Int. Conf. Image Processing (ICIP2000)*, ročník 1, Citeseer, 2000, s. 351–354.

[10] Saleeba, Z.: PicoC. 2010, [Online; 13-Března-2010].
URL http://code.google.com/p/picoc/

[11] Stoev, S.: Rafsi-a fast watershed algorithm based on rainfalling simulation. In *Proceedings of 8-th International Conference on Computer Graphics, Visualization, and Interactive Digital Media (WSCG'2000)*, Citeseer, 2000, s. 100–107.

[12] Surveyor-Corporation: Surveyor. 2010, [Online; 11-Dubna-2010].
URL http://www.surveyor.com/

Příloha A

Obsah CD

- ./robot/bin/ - složka s přeloženým firmwarem pro robota
- ./robot/complete_sources/ - složka s kompletnímy zdrojovými kódy
  ./robot/complete_sources/make.bat - dávkový soubor pro překlad kódu na platformě windows
- ./robot/console/ - složka s konzolemi pro připojení k robotu
  ./robot/console/console.bat - dávkový soubor pro spuštění grafické konzole robota
- ./proto_segmentace/ - složka s prototipovou funkci na segmentaci obrazu vytvořená ve VisualStudio C++ 2005 s podporu OpenCV
  ./proto_segmentace/release/segmentace.exe - spustitelný soubor s demonstrací segmentace
- ./manual/ - manuál k ovládání robota pomocí konzole
- ./additional/ - složka s nástroji pro kompilaci a upload firmwaru
- ./demo/ - demonstrační videa funkčnosti robota